TREEPPL - A DSL IN MIKING FOR PHYLOGENETICS

Viktor Senderov

https://vsenderov.github.io/2024-miking-workshop-treeppl

Miking Workshop 2024

KTH Campus, Digital Futures Hub, Stockholm, Sweden

4 Dec 2024

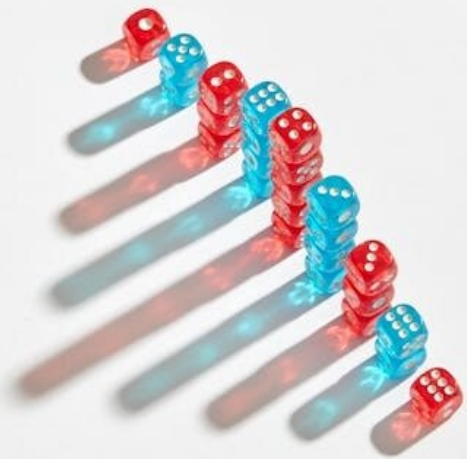
RECAP

- Phylogenetic inference
  - complex models
  - large and variable number of r.v.'s
  - hard-coded MCMC
- Challenges
  - Manual implementation of MCMC
  - performance on large data-sets
  - cannot do model comparison
  - lack of validatation
- In this presentation
  - Use Miking to develop a **probabilistic programming language** (PPL) aimed at computational biologists

PROBABILISTIC PROGRAMMING LANGUAGES

PPLS AND BAYESIAN ANALYSIS

- A probabilistic program
  - Bayesian model
  - conditional simulation
- Compilation
  - static analysis
  - statistical inference



Given an **abstract problem**,
e.g. parameter estimation of a Bayesian model. Difference between **programming** and **probabilistic programming**?

Solution,
e.g. MCMC sampling of the posterior parameters,
encoded by the programmer.

Problem, a programmatic representation of the Bayesian model as a conditional simulation,
encoded by the programmer.

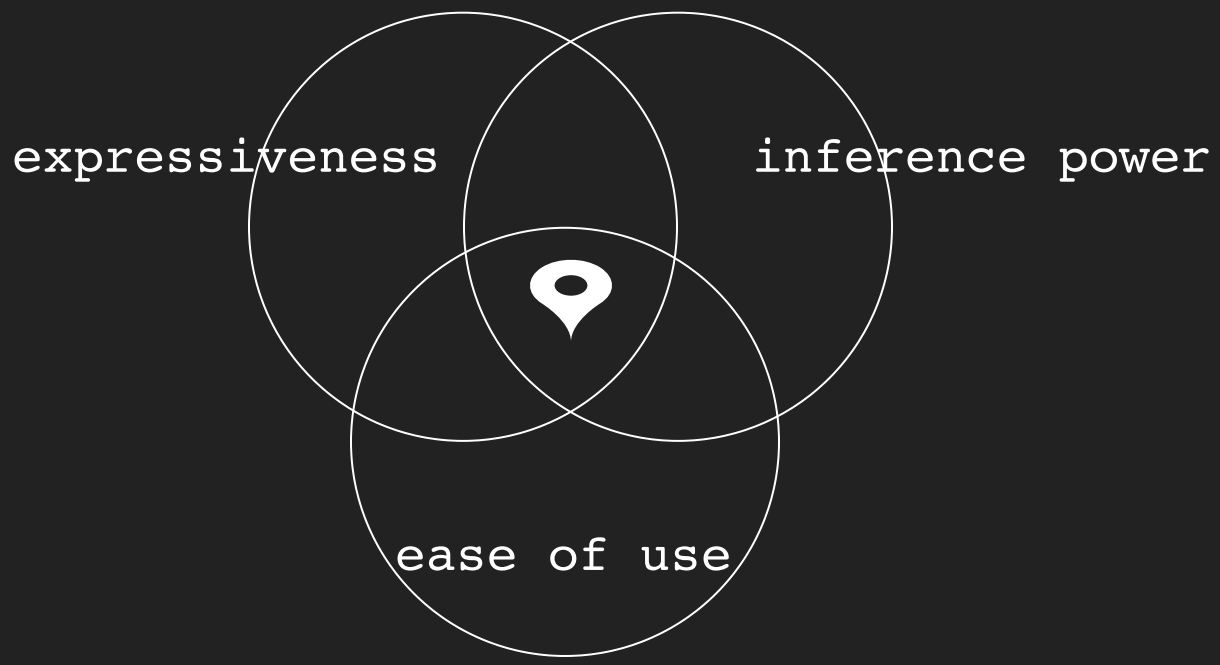
Solution, e.g. MCMC sampling of the posterior parameters,
supplied automatically by the compiler.

$$\overbrace{p(\theta|x)}^{\text{posterior distribution}} = \frac{\overbrace{p(x|\theta)}^{\substack{\texttt{observe} \\ \text{likelihood}}}\,\overbrace{p(\theta)}^{\substack{\texttt{assume} \\ \text{prior}}}}{\underbrace{p(x)}_{\text{normalizing constant}}}$$

HISTORY OF PPLS IN BIOLOGY

- 1989 - **BUGS** (Bayesian inference Using Gibbs Sampling)
- 1997 - **WinBUGS**
- 2005 - **PyMC**: MCMC-based.
- 2007 - **JAGS** (Just Another Gibbs Sampler)
- 2008 - **Church**: a Lisp-like PPL, expressiveness, recursive stochastic functions.
- 2008 - **Infer.NET**
- 2012 - **Stan**: Hamiltonian Monte Carlo (HMC)
- 2014 - **WebPPL**: a PPL that runs in web browsers.
- 2016 - **Edward** (and **Edward2**): TensorFlow
- 2020 - **Turing.jl**: A PPL for Julia

- 2016 - **RevBayes:** PGM-based
- 2023 - **LinguaPhylo**
- 2024 - **BALI-Phy**


expressiveness
inference power
ease of use

TREEPPL

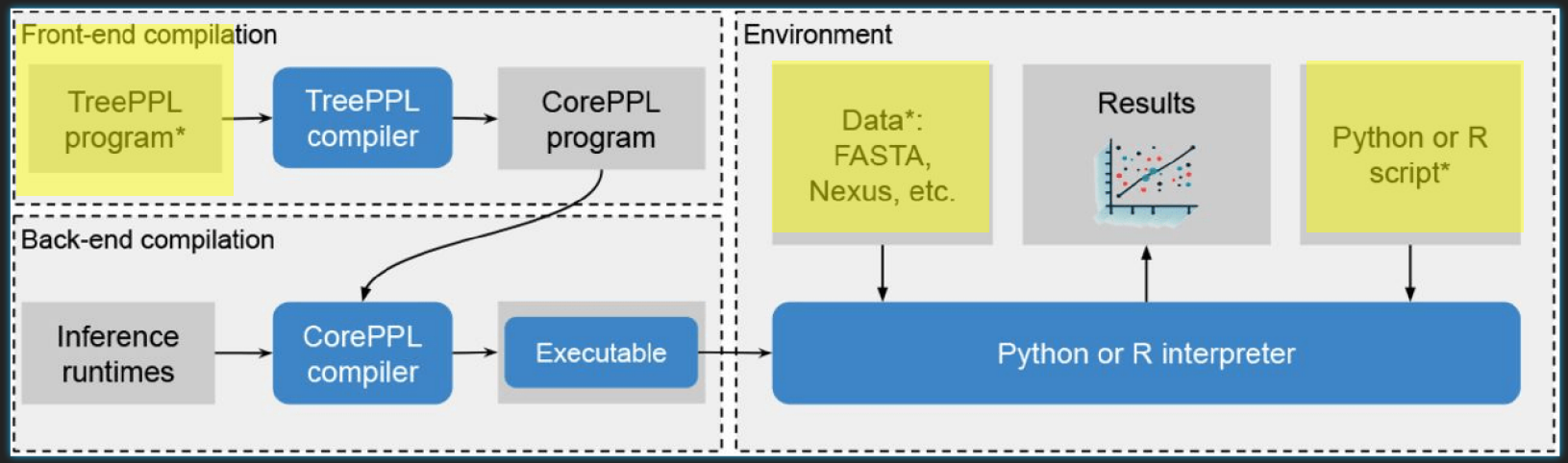
Download: https://github.com/treeppl/treeppl

Installation and documentation: https://treeppl.org

Pre-print: https://doi.org/10.1101/2023.10.10.561673

The user **models**, e.g. a regression model, expressed as a **conditional simulation** (not the inference!)...

... and supplies the **data** and the data manipulation functions.



The **inference code** is then automatically attached to the program by the compiler.

"HELLO, WORLD!"

```
function flip(datapoint: Bool, probability: Real) {
  observe datapoint ~ Bernoulli(probability);
}

model function coinModel(coinflips: Bool[]) {
  assume p ~ Beta(2.0, 2.0); // prior
  let n = length(coinflips);
  for i in 1 to n {
    flip(coinflips[i], p); // likelihood
  }
  return(p); // posterior
}
```

TYPES

- Statically typed language
- Basic datatypes: *integer, real, boolean,* and *string.*
- Vectors
- Tensors
- Records
- Sum types (a type of algebraic data type)

```
type Tree =
  | Leaf {age: Real}
  | Node {left: Tree, right: Tree, age: Real}
```

FUNCTIONS

- Can be passed as arguments to other functions
- Anonymous functions supported
- Partial application supported

```
1 model function example():() {
2   let c = 10;
3   let x = sapply(1 to 5, function (i: Int) {
4     return addi(c, i);
5   });
6 }
```

```
1 // evolveMessageClosure is a function of 4 paramters
2 let evolveMessage = evolveMessageClosure(m, t, u); // partial application
3 let ret = sapply(messages, evolveMessage);
```

DATA MODEL

- Variables are immutable
- Shadowing is allowed

```
model function shadowing():() {
    let x = 200;
    printLn(int2string(x));
    if (true) {
        let x = 300;
        printLn(int2string(x));
    }
    printLn(int2string(x));
    return;
}

// outputs 200, 300, 200
```

PROBABILISTIC PROGRAMMING

Use the `assume` keyword to introduce a random variable

```
assume p ~ Beta(a, b);
assume x ~ Exponential(rate);
assume y ~ Gamma(shape, scale);
assume w ~ Gaussian(mean, stdDev);
assume v ~ Bernoulli(prob);
```

TreePPL is a *universal PPL*

- Number of r.v. does not have to be known in advance
- R.v.'s can be defined in stochastic recursive functions
- and in stochastic branches (`if`'s)

PROBABILISTIC PROGRAMMING 1

Use the `observe` keyword to condition the likelihood on observed data

```
observe data ~ Beta(a, b);
observe data ~ Exponential(a, b);
observe data ~ Gamma(shape, scale);
observe data ~ Gaussian(shape, scale);
observe data ~ Bernoulli(prob);
```

PROBABILISTIC PROGRAMMING 2

To manipulate the likelihood directly use `weight` or `logWeight`

```
weight(lik); // lik is not on the logartihmic scale
logWeight(lik); // lik is on the logarithmic scale
```

PROBABILISTIC PROGRAMMING 3

The posterior is the returned value of the model function

```
return p; // If the likelihood was not manipulated
          // it will be exactly the prior
```

STATISTICAL INFERENCE

- importance sampling
- Various sequential Monte Carlo schemas
  - bootstrap particle filter (BPF) and the alive particle filter (APF)
- Various Markov-chain Monte Carlo schemas
  - lightweight MCMC, trace MCMC, naive MCMC
- hybrid schema: particle MCMC–particle independent Metropolis-Hastings (PMCMC-PIMH)

PATH DEGENERACY

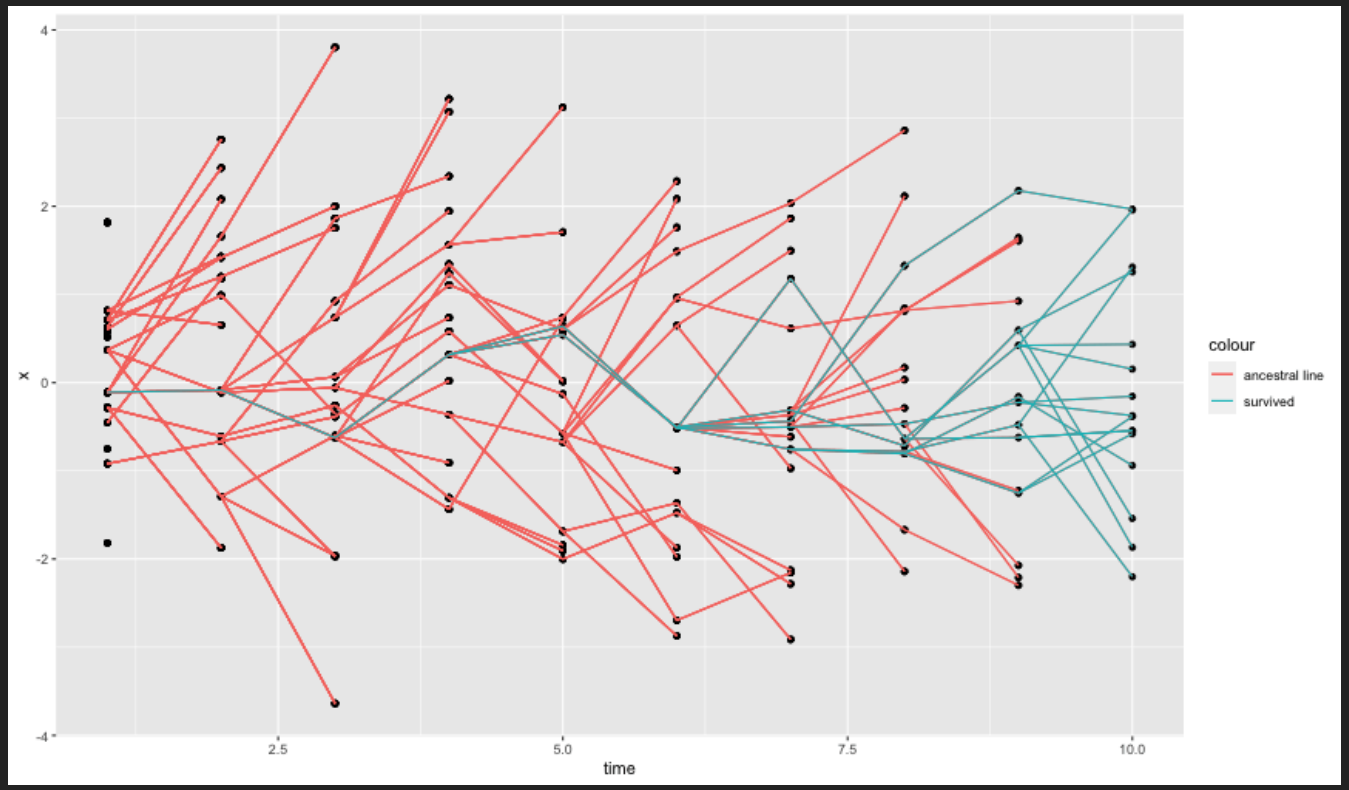


Illustration of path degeneracy

https://awllee.github.io/smc-tutorial/smc-tutorial.html#36

We offer the alive particle filter to combat that.

DELAYED SAMPLING

- A variance-minimization technique
- You don't need to sample all r.v.'s in a model

Let $k \in \mathbb{N}$ in

$$\nu \sim Gamma(k, \Theta),$$

Then, to draw from

$$n \sim Poisson(\nu t)$$

we don't need to sample ν explicitly! Instead we can

$$n \sim NegativeBionomial(k, \frac{1}{1 + t\theta})$$

and then update (backpropagate the belief) about ν to

$$\nu \sim Gamma(k + n, \tfrac{\theta}{1+t\theta}).$$

FURTHER OPTIMIZATIONS

- Alignment analysis
- Partial CPS transformation

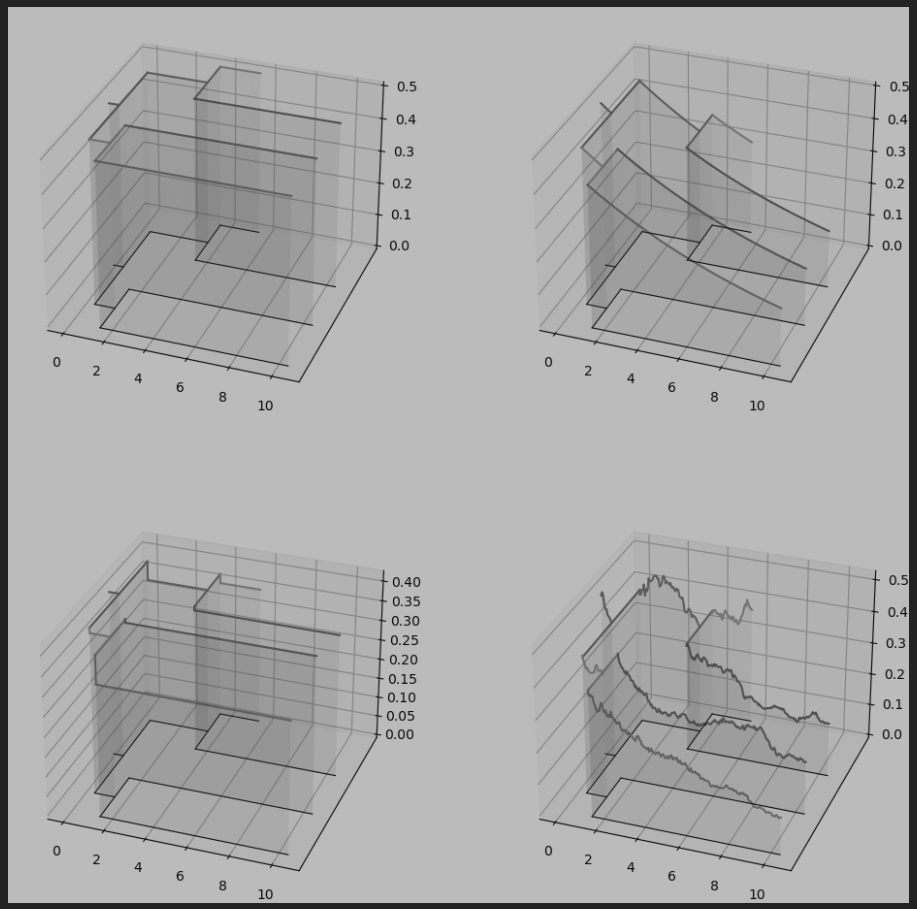
PHYLOGENTICS WITH TREEPPL

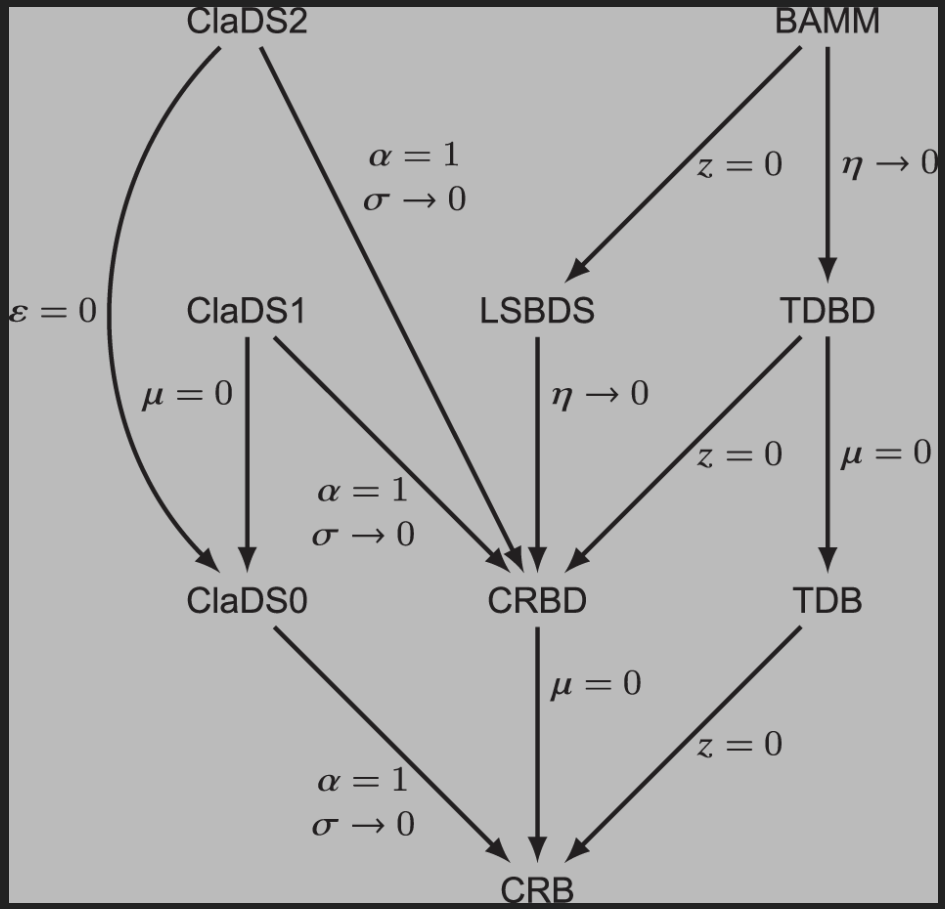
- Interfaces to Python and R to use BioPython and Bioconductor for I/O and plotting
- Tree datatypes and associated functions in standard library
- Inference methods and optimizations very useful for phylogenetic problems
- Example models

MACROEVOLUTIONARY DIVERSIFICATION MODELS

CRBD

TDBD





ClaDS

AnaDS/BDD

MESSAGE-PASSING LIKELIHOOD CALCULATION

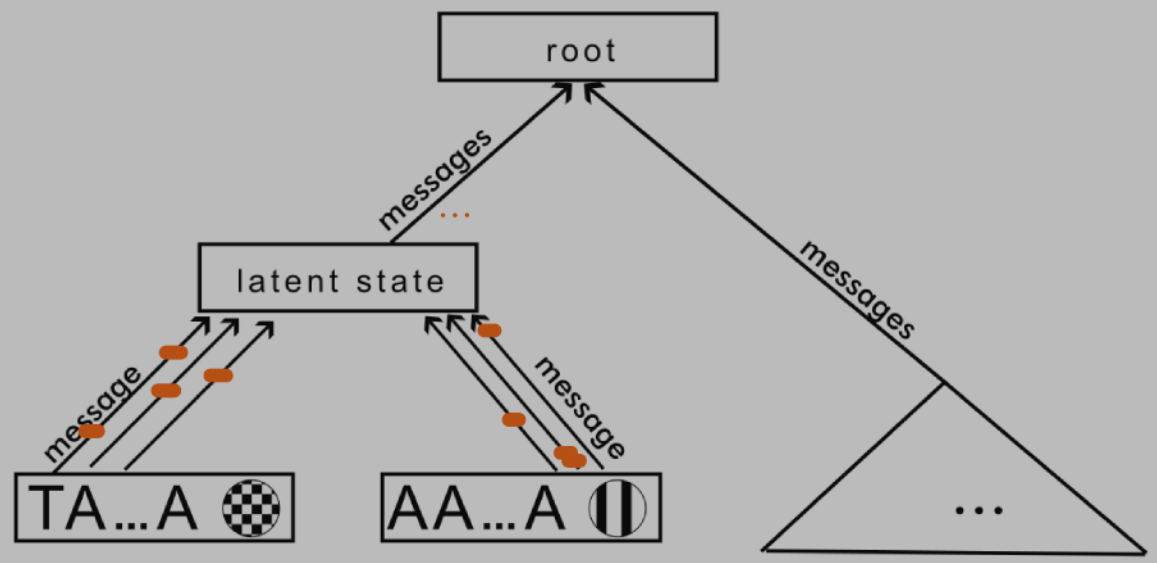
1. For a gene of length N, we compute the s-jump fraction $u \leftarrow \frac{1}{N}$.
2. For each nucleotide $i \in \{1, \dots, N\}$,
   1. Sample the number of s-jumps:

      $n_i \sim \text{Poisson}(uvt)$
   2. Compute the transition probability matrix after time t using a combination of the jump-matrix J_μ and the CTMC rate matrix Q_μ:

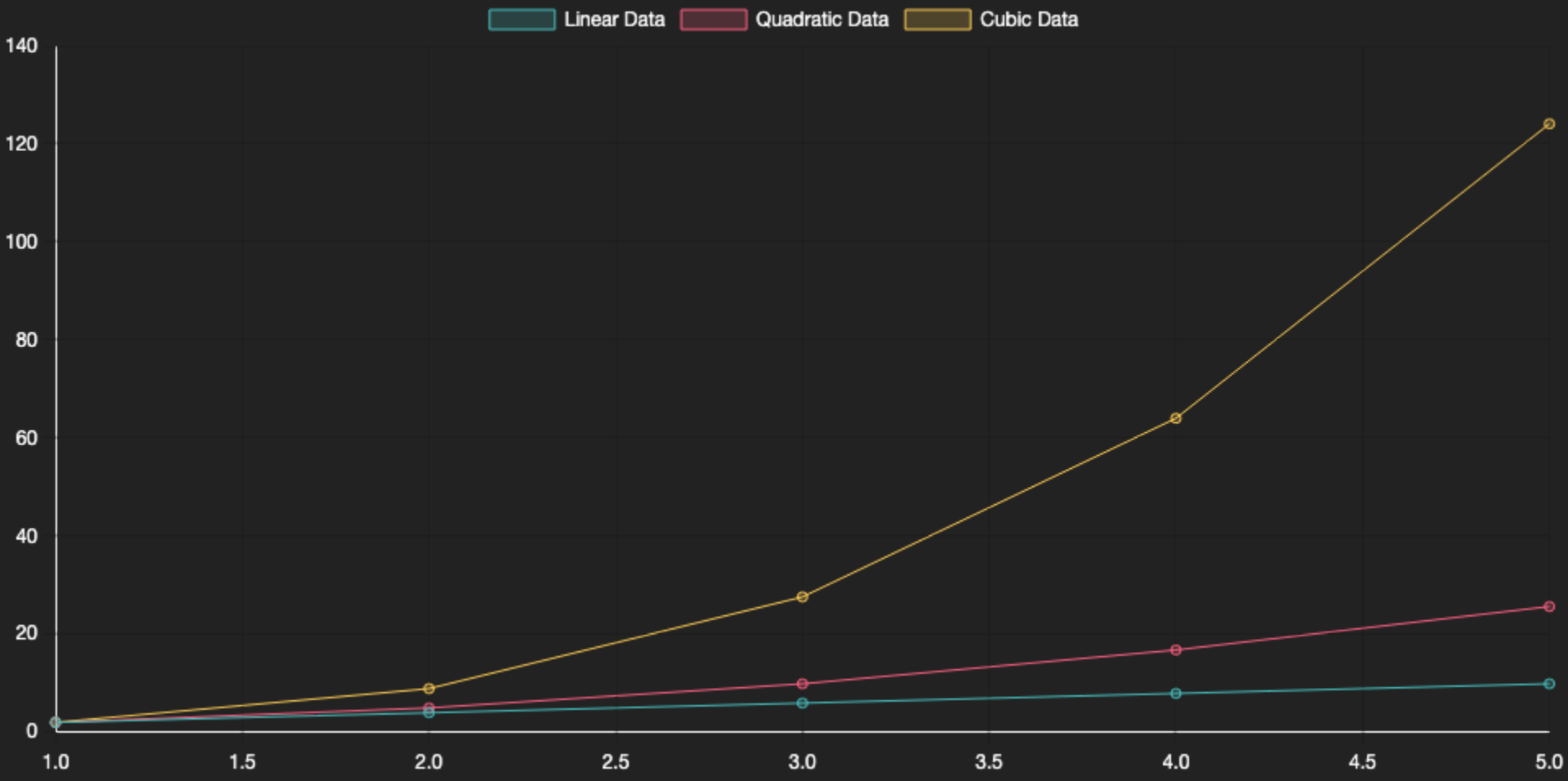
      $P_\mu^{(i)}(t) = J_\mu^{n_i} \exp(\mu t Q_\mu)$
3. Add up the s-jumps accumulated over the gene

   $n_s \leftarrow \sum_{i=1}^{N} n_i$
4. Compute the transition probability matrix for the phenotype

   $P_\lambda(t) = J_\lambda^{n_s} \exp(\lambda t Q_\lambda)$



Likelihood computation


Linear Data
Quadratic Data
Cubic Data
140
120
100
80
60
40
20
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

http://100.79.217.116:8888/notebooks/Poly.ipynb

TreePPL source code for the polynomial regression:

```
model function poly(data: Real[][]): Int {
    assume n ~ Poisson(1.0); // sample the degree of the polynomial

    // Sample the coefficients from n + 1 (we need one coeff for const poly)
    let coeffs = repApply(addi(n, 1), function() {
        assume g ~ Gaussian(0.0, 1.0);
        return g;
    });

    // likelihood
    let sigma = 1.0; // noise term
    sapply(data, function(datum: Real[]) {
        let predictedY = polynomialFunction(coeffs, datum[1]);
        observe datum[2] ~ Gaussian(predictedY, sigma);
    });

    return n;
}
```

Not shown: a function `polynomialFunction` that evaluates a polynomial as

specified by a coefficient vector at a given data point.

- TreePPL: a DSL built on top of the Miking platform
  - universal: stochastic recursion and stochastic branching
  - easy syntax
  - static types (with type inference)
  - compiled
  - supports different inference schemas
  - has a library an interface suitable for biologists

IS TREEPPL A DSL?

YES

- Inference powerful enough for phylogenetics
- Syntax based on languages familiar to phylogeneticists
- Interface to Python and R enabling the use of BioPython and Bioconductor
- Examples on website based on phylogenetic problems
- Tree datatypes in the standard library
- Developers have knowledge of phylogenetics

YES, AND ...

- SMC and MCMC schemas useful across domains
- C-like syntax familiar to empiricists across domains
- Python and R provide I/O packages across domains
- ~~Examples are not easily generalizable to e.g. finance~~
- *Tree datatypes exist in some domains*
- *Developers would not be able support user queries from e.g. physics*

SOME REFERENCES

Senderov, Kudlicka, Ronquist et al. "Universal probabilistic programming offers a powerful approach to statistical phylogenetics." Communications biology 4.1 (2021): 244.

Senderov, Kudlicka et al. "TreePPL: A Universal Probabilistic Programming Language for Phylogenetics." bioRxiv (2023): 2023-10.

Kudlicka, Jan, et al. "Probabilistic programming for birth-death models of evolution using an alive particle filter with delayed sampling." Uncertainty in Artificial Intelligence. PMLR, 2020.

Lundén, Daniel, et al. "Automatic Alignment in Higher-Order Probabilistic Programming Languages." ESOP. 2023.

THANK YOU

David Broman
Daniel Lundén
Emma Granqvist
Fredrik Ronquist
Gizem Çaylak
Jan Kudlicka
Jérémy Andréoletti
Mariana Braga
Thimothée Virgoulay
Viktor Palmkvist

Funded by the EU Horizon 2020 Marie Skłodowska-Curie grant agreement PhyPPL No. 898120, and by the Swedish Foundation for Strategic Research, as well as Vetenskapsrådet (VR).